

Script Perl que ayuda a los usuarios a superar entrevistas de trabajo

BOLOS PARA PERL

thomasstora44/Pixelio.de

Ya casi nadie escribe algoritmos complicados, pero los tests de aptitud a menudo requieren que los candidatos tengan la teoría bajo control.

POR MICHAEL SCHILLI

El final de la crisis está cercano. ¡Es hora de buscar un nuevo empleo! Una vez que la economía vuelva a levantarse, las empresas volverán a tener dinero para gastar. Y, en el paraíso de los trabajos de ensueño, los candidatos tendrán que sacudir el polvo de sus CVs y empollar las preguntas a superar en las entrevistas de trabajo.

Una Bola que Cae

Una pregunta muy popular que a las compañías de software de Silicon Valley les

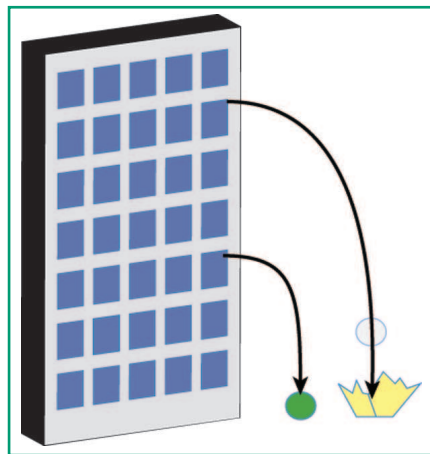


Figura 1: ¿Desde qué planta sobrevivirá la bola? Dadas 100 plantas y un total de dos bolas, será suficiente con un máximo de 15 intentos.

encanta hacer a los candidatos a puestos de programador es ésta: dadas dos bolas idénticas de bolos, ¿desde qué planta de un edificio de 100 plantas podríamos soltar cualquiera de ellas sin romperlas? La idea de esta tarea es minimizar el número de intentos de lanzamiento necesarios en el peor caso.

Un método muy simple sería empezar en la primera planta, lanzar la bola, y ver si sobrevive. Si lo hace, repetiríamos el experimento en la segunda planta y trabajaríamos lentamente hasta la planta número 100. En el peor caso, es decir, si las bolas fueran suficientemente duras como para resistir un lanzamiento desde la planta 100, necesitaríamos 100 lanzamientos para encontrar el resultado.

Recordemos que tenemos dos bolas de bolos, y que se nos permite romper ambas, aunque tendríamos que identificar la planta crítica precisamente cuando rompamos la segunda bola. (Pista: nos debería llevar 15 intentos, a menos que queramos suspender el test de aptitud y tengamos que buscar un trabajo en otra parte).

Si piensa que tiene lo que hay que tener para ser un desarrollador de software en Silicon Valley, deje de leer en este momento, imagine un escenario libre de estrés en una entrevista de trabajo y mantenga la cabeza fría mientras sopesa las opciones que tiene en mente. El reloj ya está contando...

El Truco Decisivo

Como mencioné previamente, un método lineal resolvería este problema, pero nos llevaría demasiadas iteraciones. Una búsqueda binaria sería similar: si dividimos las 100 plantas entre dos y lanzamos la primera bola desde la planta 50, necesitaríamos 49 pasos más para encontrar la planta crítica en el peor de los casos. Después de todo, no se nos permite romper la segunda bola, si no tenemos la respuesta acto seguido.

El truco es dividir las 100 plantas en tramos de tamaño decreciente (véase la Figura 1). El primer tramo tiene una longitud de n plantas, el segundo de $n-1$, el tercero de $n-2$, y así hasta un tramo que incluya la planta 100. Usando esta división, iríamos pasando por los tramos de izquierda a derecha, lanzando la primera bola desde la primera planta del tramo en el que estamos actualmente trabajando. Si se rompe la bola, deberíamos arrojar la segunda bola desde la segunda planta del tramo anterior

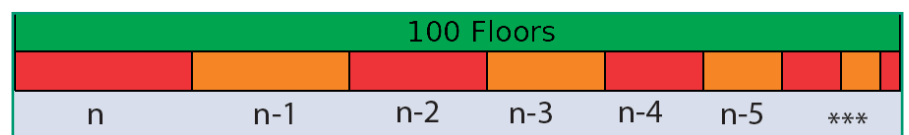


Figura 2: Dividimos las 100 plantas en tramos de tamaño decreciente.

$$\frac{n(n+1)}{2} \geq 100$$

$$n^2 + n - 200 \geq 0$$

$$n_{1/2} = \frac{-1 \pm \sqrt{1^2 + 4 \cdot 1 \cdot 200}}{2} = 13.65, -14.65$$

Figura 3: La condición se vuelve una ecuación cuadrática que puede resolverse mediante la conocida fórmula [2].

(si existe) hasta que se rompa, o hasta que alcancemos el final del tramo. Esto identifica la planta crítica antes de que nos quedemos sin bolas.

La propiedad interesante acerca de la división mostrada en la Figura 2 es que siempre limita el número de intentos para identificar la planta crítica a un máximo de $n + 1$, sin importar dónde se rompa finalmente la bola. Si el punto crítico fuese, pongamos por caso, la primera planta del primer tramo, la primera bola se rompería en el segundo lanzamiento teniendo todo en cuenta, desde la primera planta del segundo tramo. Tras esto, el investigador lanzaría la segunda bola desde la segunda planta hasta la planta n de la primera sección. Sobreviviría hasta el final, dándonos n como la planta más alta. El número de intentos aquí es dos con la primera bola, y

100 Floors					
1-14	15-27	...	85-90	91-95	96-99 100

Figura 4: Cuando $n = 14$, las plantas se dividen en estos tramos.

$n-1$ con la segunda, dándonos un total de $n + 1$.

Si la planta crítica es la última del quinto tramo, que tiene una longitud de $n-4$, la primera bola se rompería en el sexto intento, desde la primera planta del sexto tramo. Entonces, sólo necesitaríamos $n-4-1$ intentos para probar con las plantas restantes en el quinto tramo, ya que este tramo tiene una longitud de $n-4$ y previamente hemos probado la primera planta sin romper la bola. El máximo número de intentos si la última bola se lanza bien desde la última planta del quinto tramo es $6 + n - 4 - 1$ (es decir, $n + 1$).

La Búsqueda de n

El algoritmo es perfectamente claro y da con el resultado en $n + 1$ intentos en el peor de los casos. Pero, ¿cómo decidimos el valor de n para un edificio con 100 plantas? La tarea es reducir el máximo número de intentos a un mínimo, por lo que esto significa que se prefieren los valores pequeños de n . Sin embargo, el tamaño de los tramos

comienza en n y luego cae en pasos de 1, y si un tramo tiene un valor de 1, sin que la suma total de todos los tramos llegue a 100, nos hemos quedado cortos.

Por tanto, la suma total de $n + (n-1) + (n-2) + \dots + 1$ debería ser precisamente 100 si fuese posible. Puede estar por encima de 100, pero no por debajo. Como seguramente sabe, se dice que el matemático alemán Carl Friedrich Gauss definió una fórmula para las series de números cuando aún era un niño. Sus profesores le pidieron que sumara los números entre 1 y 100, y el genial Gauss les contestó en cuestión de segundos sin hacer una sola suma [1].

Por tanto, de acuerdo a Gauss, $n(n + 1)/2 > = 100$. Esto conduce a una ecuación cuadrática, y la única solución positiva posible, según la fórmula matemática que nos enseñaron en la escuela, es 13,65. La Figura 3 muestra cómo la he resuelto en mi pizarra. El valor mínimo aceptable para n es por tanto 14; la Figura 4 muestra la distribución derivada de los tramos de plantas que estábamos buscando.

Listado 1: bball-drop

```
01#!/usr/local/bin/perl -w
02use strict;
03use POSIX qw(ceil);
04use Log::Log4perl qw(:easy);
05Log::Log4perl->
    easy_init(INFO);
06
07my $total = 100;
08
09my $n = ceil((-1 +
    sqrt(1+4*2*$total))
10    / 2);
11
12my $sum = 1;
13my @stops = ();
14push @stops, $sum;
15
16while($sum + $n <= $total) {
17    push @stops, $sum + $n;
18    $sum += $n;
19    $n--;
20}
21
22my $last_ok_floor =
23    (defined ARGV[0] ? ARGV[0] :
    42);
24
25INFO "Pst, pst: Highest OK floor
    is ";
26    $last_ok_floor;
27
28my $tries = 0;
29my $ok_floor = 1;
30my $smash_floor = $total + 1;
31
32for my $stop (@stops) {
33    $tries++;
34    if(!try_floor($stop,
    $last_ok_floor)) {
35        $smash_floor = $stop;
36        last;
37    }
38    $ok_floor = $stop;
39}
40
41for my $try_floor ($ok_floor+1
    ..
42    $smash_floor-1) {
43    $tries++;
44    if(!try_floor($try_floor,
    $last_ok_floor)) {
45        $smash_floor = $try_floor;
46        last;
47    }
48}
49    $smash_floor = $try_floor + 1;
50}
51
52INFO "Highest OK floor: ",
    $smash_floor-1,
53    " ($tries tries)";
54
55#####
56sub try_floor {
57    #####
58    my($floor, $last_ok_floor) =
    @_;
59
60    if($floor > $last_ok_floor) {
61        INFO "Floor $floor: Wham,
    busted!";
62        return 0;
63    }
64
65    INFO "Floor $floor: Okay.";
66    return 1;
67}
```

```

$ bball-drop 14
2009/10/10 13:36:05 Pst, pst: Highest OK floor is 14
2009/10/10 13:36:05 Floor 1: Okay.
2009/10/10 13:36:05 Floor 15: Wham, busted!
2009/10/10 13:36:05 Floor 2: Okay.
2009/10/10 13:36:05 Floor 3: Okay.
2009/10/10 13:36:05 Floor 4: Okay.
2009/10/10 13:36:05 Floor 5: Okay.
2009/10/10 13:36:05 Floor 6: Okay.
2009/10/10 13:36:05 Floor 7: Okay.
2009/10/10 13:36:05 Floor 8: Okay.
2009/10/10 13:36:05 Floor 9: Okay.
2009/10/10 13:36:05 Floor 10: Okay.
2009/10/10 13:36:05 Floor 11: Okay.
2009/10/10 13:36:05 Floor 12: Okay.
2009/10/10 13:36:05 Floor 13: Okay.
2009/10/10 13:36:05 Floor 14: Okay.
2009/10/10 13:36:05 Highest OK floor: 14 (15 tries)
$

```

Figura 5: El peor caso: el algoritmo necesita 15 pasos.

Moldeado en Perl

El programa `bball-drop` del Listado 1 muestra una implementación del algoritmo en Perl [3]. La línea 9 determina el valor de n resolviendo la ecuación cuadrática y redondea el resultado en punto flotante al siguiente entero con la función `ceil()` del módulo `POSIX`.

El bucle `while` de las líneas 16 a 20 crea entonces un array, `@stops`, cuyos elementos se fijan al número de la primera planta de un tramo particular. Es decir, para $n = 14$, `@stops` contiene los valores 1, 15, 28, ..., 91, 96 y 100. El script llama a la última planta desde la que la bola ha sobrevivido como `Highest OK floor` y acepta este número en la línea de comandos para probar (llamaríamos al script tecleando `bball-drop 99` para probar el escenario en el cual las bolas pueden sobrevivir a la planta 99, por ejemplo, pero no más arriba), o toma el valor por defecto de 42 fijado en la línea 23.

`Log4perl` en la línea 25 le pasa discretamente el valor a descubrir por el algoritmo (`Pst,pst`) al llamante, y luego el script comienza a trabajar con los casos de

prueba individual. El bucle `for` de las líneas 32 a 39 se dirige al comienzo de un tramo en el array `@stop` y llama a la rutina `try_floor()` con la planta a probar y la planta máxima secreta como argumentos. Si la altura probada es mayor que la altura máxima, `try_floor()` devuelve falso para indicar

que la bola que hemos tirado se ha deshecho en miles de pedazos.

Si el bucle `for` de las líneas 32 a 39 encuentran una planta desde la que la primera bola no sobrevive, finaliza haciendo una llamada a `last` y fija la variable `$smash_floor` a la planta desde la que se ha causado el destrozo. El algoritmo continúa con el siguiente bucle `for` en las líneas 41 a 50, tomando la siguiente planta del tramo con el último lanzamiento exitoso y subiendo una planta cada vez hasta que la segunda bola se haga añicos, o hasta que se llegue al final del tramo. Si la bola se rompe, el bucle termina con `last`, y el resultado está disponible en `$smash_floor`. Reduciendo este valor en 1 nos da la mayor planta desde la cual la bola sobreviviría al lanzamiento. Si se alcanza el final del tramo, la última planta exitosa es el máximo buscado.

¡Compruebe los Resultados!

Con problemas delicados como éste, los bugs son casi inevitables, por lo que será mejor verificar los resultados con un conjunto de tests de regresión. El programa

```

$ prove ./suite
./suite....ok
All tests successful.
Files=1, Tests=202, 7 wallclock secs
( 0.06 usr 0.02 sys + 5.31 cusr
 1.12 csys = 6.51 CPU)
Result: PASS
$

```

Figura 6: La suite de tests confirma que el script devuelve correctamente los resultados para cualquier combinación de plantas y nunca toma más de 15 intentos.

`suite` (Listado 2) usa el módulo `Sysadm::Install` de CPAN para llamar al script `bball-drop` repetidamente con diferentes valores de planta máxima entre 0 y 100. La función `tap()` llama al script y captura `STDOUT`, `STDERR` y el código devuelto por el script. En una ejecución típica, el script `bball-drop` muestra algo como lo que presenta la Figura 5, y la expresión regular de la línea 14 toma el resultado con la planta más alta desde la que ha sobrevivido la bola y el número total de pasos requeridos para averiguar el resultado.

Los dos comandos `Test::More` de las líneas 19 y 22 verifican si el resultado identificado por el script coincide con el parámetro predefinido y si el script mantiene el número máximo permitido de 15 intentos. El módulo `Test::More` de CPAN proporciona una salida muy probada en formato TAP (1 `ok` en los casos exitosos, 2 `not ok` para los no satisfactorios). Leer esto a simple vista puede ser engorroso, por lo que el script `prove`, que viene con el módulo y con las distribuciones más recientes de Perl, engloba un conjunto de tests alrededor de las salidas TAP para confirmar que los 202 tests se completaron exitosamente (véase la Figura 6). Esto es mucho más fácil que verificar manualmente las más de 200 líneas de salida de `suite` en busca de errores. Al final, los tests se pasan con facilidad. ¡El candidato obtiene todos los puntos y afronta por delante una exitosa carrera en el mundo TIC!

Listado 2: suite

```

01 #!/usr/local/bin/perl -w          16 my($result, $tries) =
02 use strict;                        17 ($1, $2);
03 use Sysadm::Install qw(:all);      18
04 use Test::More;                    19 is($floor, $result,
05                                     20 "result: $result $tries");
06 plan tests => 202;                 21
07                                     22 ok(
08 for my $floor (0..100) {           23 $tries <= 15,
09                                     24 "result: $result $tries"
10 my($stdout, $stderr, $rc) =        25 );
11 tap("bball-drop", $floor);         26
12                                     27 } else {
13 if($stderr =~                       28 die "Unmatched: $stderr";
14 /floor: (\d+) \((\d+)\)/) {        29 }
15                                     30 }

```

RECURSOS

- [1] Como el joven Johann Carl Friedrich Gauss resolvió la suma de enteros en una progresión aritmética: http://en.wikipedia.org/wiki/Gauss#Early_years_.281777.E2.80.931798.29
- [2] Ecuación cuadrática: http://en.wikipedia.org/wiki/Quadratic_equation
- [3] Listados de este artículo: <http://www.linux-magazine.es/Magazine/Downloads/62>